目錄

- Introduction 1.1
- 入门教程 1.2
  - hello world 1.2.1
  - 数值、字符与字符串 1.2.2
  - 运算符及表达式 1.2.3
  - 选择结构与循环结构 1.2.4
  - 函数基本概念及作用域 1.2.5
  - 编译预处理命令 1.2.6
  - 数组 1.2.7
  - 指针的基本概念 1.2.8
  - 指针与数组 1.2.9
  - 字符串处理 1.2.10
  - 调试以及资料收集 1.2.11
  - 结构体 1.2.12
- 数学基础 1.3
  - 信息存储 1.3.1
  - CPU如何进行数学运算 1.3.2

C语言基础教程

> 作者：Lellansin
>
> 来源：C-basic-tutorial

- 入门教程
    - hello world
    - 数值、字符与字符串
    - 运算符及表达式
    - 选择结构与循环结构
    - 函数基本概念及作用域
    - 编译预处理命令
    - 数组
    - 指针的基本概念
    - 指针与数组
    - 字符串处理
    - 调试以及资料收集
    - 结构体
- 数学基础
    - 信息存储
    - CPU如何进行数学运算

教程预览地址：http://doc.lellansin.com

Note: 本教程使用 GitBook 生成。观看页面如有BUG或者想要改进请访问 GitBook on GitHub.

Hello world

压缩版下载地址：vc6_cn_setup 完整版、加强版等下载地址：Visual C++ 6.0

1.下载 vc6_cn_setup.exe 或者是其他版本的vc6 （推荐下载完整版，或者不嫌麻烦的可以去下载 visual studio 2010 或 2012） 2.安装vc 6.0 （一路确定过去就行了） 3.打开安装好的vc 6.0，如果操作系统是win7的话会提示不兼容，这个可以直接忽略，直接运行程序就可以了。

关掉每日提示，文件->新建->切换到文件->C++ source file->确定

到这里，我们可以先保存一下，保存的时候，劲量新建一个文件夹，保存在新文件夹内。避免编译之类的操作新生成一些文件，使得文件看起来很乱。

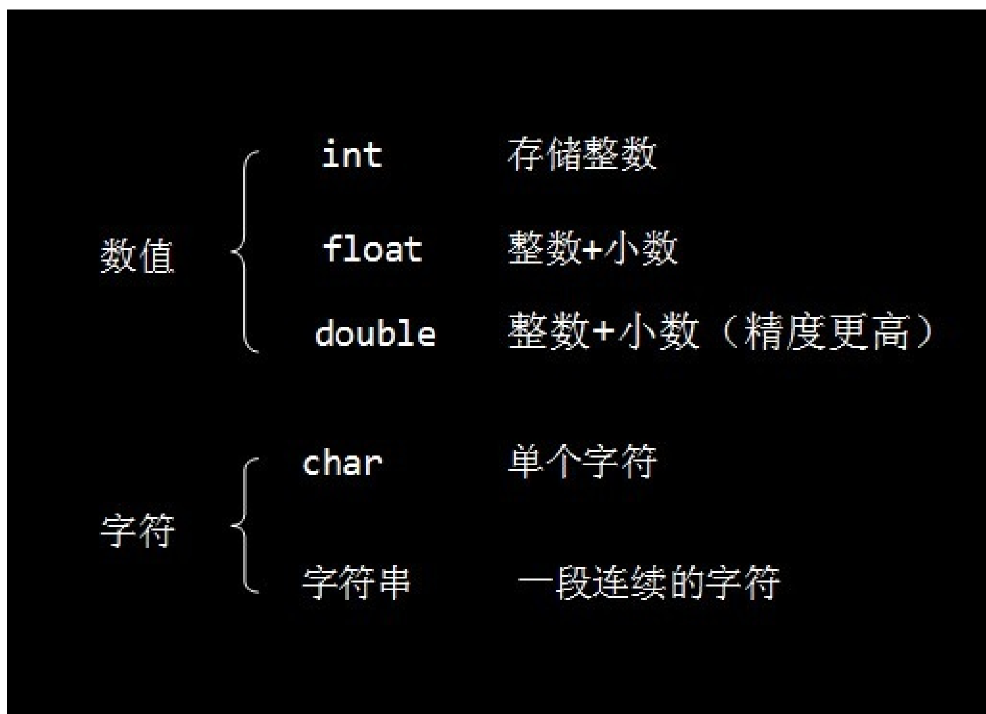
接下来开始输入第一个程序的代码：

```
#include<stdio.h>
int main()
{
    printf("Hello world");
}
```

输入完后记得保存（快捷键ctrl+s），然后找到菜单上的 组建->编译， 组建->执行

如果运行成功，那么恭喜你，你已经开始了你的编程之旅 如果有报错，可以仔细核对一下你的代码，看看是不是拼写有错

数值、字符与字符串



声明方式

数据类型 变量名; 举例： ```cpp // int类型 int i; /* 普通声明 */ int j,k; /* 同时声明多个 */ int age = 18; /* 声明的同时赋值 */ int Alan ,Sam=16;/* 声明与同时赋值 */ // float类型 float f; /* 普通声明 */ float q,money; /* 同时声明多个 */ float v = 2.0; /* 声明的同时赋值 */ // char类型 char c; /* 普通声明 */ char zh,text; /* 同时声明多个 */ char letter = 'A'; /* 声明的同时赋值 */ // char数组(字符串) char c[20] = { 'H','e','l','l','o',' ','w','o','r','l','d' }; char name[] = { 'A', 'l', 'a', 'n' }; ```

转义字符

在一段字符串中，不能直接出现，需要转义的字符，例如：

'\n' 换行 '\t" 水平制表 '\" 单引号 '\"' 双引号 '\' 反斜杠

```
#include<stdio.h>
main()
{
    printf("Num\tName\n");
    printf("001\tAlan\n");
    printf("002\tLellansin\n");
}
```

```
输出结果：
Num     Name
001     Alan
002     Lellansin
```

printf与scanf

printf 打印，scanf 为输入

形式

```
printf(字符串，... );  // 点点点为多个参数
scanf(字符串，... );
```

占位符

对于 printf 输出的情况而言，占位符的意思是先占个位置的符号，至于这个位置将来会放什么还不确定，只能先确定是占位符所指明的类型，具体的值还要等稍后的参数中补充。对于 scanf 来说，则是相反的情况。

常见格式：

%d 十进制整数 %i 十进制整数 %c 单个字符 %s 字符串 %f 浮点数

如：

```
// 输出字符串要输出的年龄尚未确实值知道是 %d 十进制整数类型,
// 具体输出来是什么值则看后面 age 的具体值
printf("Alan的年龄是%d", age);

// 接手用户输入的一个十进制整数，注意输入是要加 & 符号
// 如果用户输入 10.2 则存入变量i中的值是 10，因为格式是十进制整数
scanf("%d", &i);
```

举例：

```
#include<stdio.h>
int main()
{
    int age;
    printf("你的年龄是？\n");
    scanf("%d",&age); // 输入时变量前加上 “&” 符号
    printf("你的年龄是%d",age);
}
```

运算符以及表达式

```c
// 算术运算符

#include <stdio.h>
int main()
{
    int i = 7, j = 8;

    printf("i+j=%d\n", i + j ); //15
    printf("i-j=%d\n", i - j ); //0
    printf("i*j=%d\n", i * j ); //56
    printf("i/j=%d\n", i / j ); //0
    printf("i%%j=%d\n", i % j );//7

    i++;
    printf("i++后i的值为%d\n",i);  //8
    printf("++i后i的值为%d\n",++i);//9

    printf("j--后j的值为%d\n",j--);//8
    printf("--j后j的值为%d\n",--j);//6
}
```

```c
// 关系与逻辑运算符

#include <stdio.h>
int main()
{
    int Alan = 18, Sam = 18, Jack = 16;

    printf("%d\n", Alan > Jack ); // 1
    printf("%d\n", Sam  < Jack ); // 0
    printf("%d\n", Alan != Sam ); // 0
    printf("%d\n", Alan == Sam ); // 1
    printf("%d\n", Alan >= Sam ); // 1
    printf("%d\n", (Sam>Jack) && (Sam>Jack) ); //1
    printf("%d\n", Alan > Jack );//1
    printf("%d\n", ! Sam );    //0
}
```

```
/*
    数据类型转换
    形式：
    (类型) (表达式)
*/

#include <stdio.h>
int main()
{
    int chinese, math, english, sum;

    printf("请输入语数英三科的成绩：(以空格隔开)\n");
    scanf("%d%d%d", &chinese, &math, &english);

    sum = (chinese + math + english);

    printf("平均分为%d\n", sum/3);
    printf("平均分为%f\n", (float)sum/3);
}
```

需要注意的是“=”与“==”之间的区别：“=”是赋值运算 “==”是逻辑运算

```
#include <stdio.h>
int main()
{
    int i = 5, b = 10;

    printf(" a=b 值为 %d\n", a = b ); //10
    printf("a==b 值为 %d\n", a == b); //0

}
```

选择结构与循环结构

选择结构

```
1.if() else
2.switch() case:
```

循环结构

```
1.while() 循环
2.do while() 循环
3.for() 循环
```

终止循环： 1.continue 终止该次循环 2.break 终止该循环

if else

```
// 形式1
if(条件)
{ /*条件为真执行*/
    ...
}

// 形式2
if(条件)
{ /*条件为真执行*/
    ...
}else
{ /*条件为假执行*/
    ...
}

// 更多形式..
if(条件1)
{ /*条件1为真执行*/
    ...
}else if(条件2)
{ /*条件2为真执行*/
    ...
}else
{ /*条件1、2为假才执行*/
    ...
}
```

代码示例：

```
#include <stdio.h>
int main()
{
    int salary = 2500;
    int rent = 850;
    int life_cost = 1000;

    if( salary > (rent + life_cost) )
    {
        printf("还过的下去");
    }else if( salary < (rent + life_cost) )
    {
        printf("活不下去了！");
    }else
    {
        printf("妈的！");
    }
}
```

switch

```
switch(变量)
{
    case 情况1 :
        ...
    break;

    case 情况2 :
        ...
    break;

    case 情况3 :
        ...
    break;

    default :  /* 默认情况 */
        ...
}
```

代码示例

```
#include <stdio.h>
int main()
{
    char cmd;

    printf("确认要删除吗？");
    scanf("%c", cmd);

    switch(cmd)
    {
        case 'Y':
            printf("文件正在删除\n");
        break;
        case 'N':
            printf("取消删除\n");
        break;
        defualt:
            printf("用户未响应，操作取消\n");
    }
}
```

关于case穿透

```
#include <stdio.h>
int main()
{
    int score = 80;

    switch(math/10)
    {
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
            printf("不及格\n");
        case 6:
        case 7:
        case 8:
            printf("及格\n");
        case 9:
        case 10:
            printf("满分\n");
    }
}
```

while 循环

```
#include <stdio.h>
int main()
{
    int i = 0;
    while(i < 10)
    {
        printf("%d ", i);
    }
}
```

do while 循环

```
#include <stdio.h>
int main()
{
    char c;

    do
    {
        printf("请问你是⑨吗？(Y/N)\n")
        scanf("%c", &c);
    }while( c != 'Y');

}
```

for 循环

```
// 形式
for( 初始值; 条件; 变化 )
{
    ...
}
```

示例:

```
#include <stdio.h>
int main()
{
    int i;
    for( i = 0; i < 10; i++ )
    {
        printf("%d\n", i);
    }
}
```

continue与break

终止循环： 1.continue 终止该次循环 2.break 终止该循环

```
#include
main()
{
    int i;
    for( i = 0; i < 10; i++)
    {
        if( i == 3 )
            continue;
        printf("%d ", i);
    }
}
```

函数的基本概念



```
// 初中
y = 3x + 2

// 高中
f(x) = 3x + 2

// c语言
int f(int x)
{
    return x * 3 + 2;
}
```

C语言中的函数

1.函数名 2.参数 3.返回值

```
#include <stdio.h>

/*
    sum函数名 返回值为int
    int a, int b 为sum的参数及其类型
    返回值为 ( a + b ) 这个表达式的值
*/
int sum(int a, int b)
{
    return a + b;
}
int main()
{
    int i, j;
    printf("请输入两个数：\n");
    scanf("%d%d", &i, &j);
    printf("%d\n", sum(i, j)); // sum(参数,参数)的形式调用,返回两数之和
}
```

函数的作用域

```
#include <stdio.h>

void swap(int i, int j)
{
    int temp;
    temp = i; i = j; j = temp;
    printf("x=%d, y=%d", x, y);
}
int main()
{
    int i = 30, j = 75;
    swap(i, j);
    printf("i=%d, j=%d\n", i, j),
}
```

int main()中的 i, j 与 swap() 中的 i,j 不是相同的变量 两个函数,是两个不同的内存块.两个变量名称相同,但不在同一个内存上 所以,如果其中某个变量有改变与另一个函数中的变量无关

编译预处理

预处理指令

include 引用各种函数库

define 定义常用的宏

条件编译

根据表达式的结果来决定要编译的内容

if

else

elif

endif

// 根据变量或宏等是否定义来决定要编译的内容

ifdef

ifndef

#include

```
// 1)两种形式:
#include <stdio.h>  // 到配置目录中找
#include "stdio.h"  // 从当前目录开始找,无则到配置目录中找

// 2)文件包含允许嵌套
// 文件A包含了文件B,文件B又包含了文件C,最终文件B和C都会被引入A文件
```

#define

宏定义，用一个标示符来表示一个字符串，这个字符串可以使常量、变量或表达式。在宏调用中，将用该字符串替换宏名。 ```cpp #include // 1) 无参数宏 #define PI 3.14 #define M (a+b+c)/2.0 int main() {   int a = 1, b = 2, c = 3;   printf("面积为%f", PI*2*2); printf("%f", M); } ``` ```cpp // 2)带参数宏 #define f(x) x*x+3*x #define Sum(a,b,c) a=b*c;b=c*a;a=b*c printf("%d", 3*f(2)); ``` PS:非值传递,而是传递形参字符

条件编译

if (常量表达式) =====> 注意常量,非变量

仅当表达式为真,则编译它与 #endif 之间的代码

```
#include<stdio.h>

#define debug = 0
#define beta = 1
#define status = 1
int main()
{
#if (status == debug)
    printf("程序调试中\n");
#elif (status == beta)
    printf("程序测试中\n");
#else
    printf("欢迎使用正式版!\n");
#endif
}
```

ifdef 如果有定义 (if define)

ifndef 如果没定义 (if not define)

```
#ifndef PI
    define PI 3.14
#endif
```

ifdef 和 #ifndef 主要是用来避免重定义

c语言入门教程 第7讲 数组

额外内容： 使用CL.exe编译c程序

数组

一维数组

同类元素的集合，称为数组

定义方式

数据类型 数组名 [常量表达式]; 举例：

```
int     a[5];
float    score[8];
double    d[10];
char    name[256];
```

初始化

```

int a[3];
a[0] = 1;
a[1] = 2;
a[2] = 3;

int b[5] = { 1, 2, 3, 4, 5 };     // 省略写法

float score[8] = { 1.0, 1.2 };    // 后面未赋值则为零

char name[] = "Alan";             // 省略写法，单独定义时必须声明大小
```

一维数组的下标

数组的下标实际上是关于数组第一个元素的偏移量 比如，定义一个数组array[5]。它的第一个元素跟数组的第一个元素的偏移量为0 所以，数组的第一个元素应该是array[0] 数组的第二个元素跟数组的第一个元素偏移量为1 所以，数组的第二个元素应该是array[1] 以此类推,数组的第五个元素与第一个元素的偏移量为4 所以，数组array的第五个元素应该是array[4] 结论：一位数组最后一个元素的下标为 (长度 - 1) 并且对于一个数组它的第几个元素的下标就是几减一

循环初始化：

```
#include <stdio.h>
int main()
{
    int a[10], b[3], i;

    // 有规律的循环赋值
    for( i = 0; i < 10; i++ )
    {
        a[i] = i;
    }

    // 多次输入赋值
    for( i = 0; i < 3; i++ )
    {
        scanf("%d", b[i]);
    }
}
```

注意如果一次输入多个数用scanf循环去接收的话，空格也会被当做字符（或数值）来保存

一维数组遍历

```
#include <stdio.h>
int main()
{
    int a[8] = { 1, 1, 2, 3, 5, 8, 13, 21 };
    int i;

    for( i = 0; i < 8; i++ )
    {
        printf("%d ", a[i]);
    }
}
```

一维数组应用示例

```
#include <stdio.h>
int main()
{
    int a[8] = { 80, 90, 85, 84, 70, 76, 75, 83 };
    int i, max = 0, min = 100;

    for( i = 0; i < 8; i++ )
    {
        if( a[i] > max )
        {
            max = a[i];
        }else if( a[i] < min )
        {
            min = a[i];
        }
    }
    printf("最高分为：%d 最低分为：%d", max, min);
}
```

二维数组

定义

```
//数据类型 数组名 [常量表达式][常量表达式];

int     a[56][3];
char    name[5][256];
```

初始化

```
// 1)分段赋值
int grade[2][3] = { {80,75,92}, {61,65,71} };

// 2)连续赋值
int a[2][3] = { 80, 75, 92, 61, 65, 71 };

// 3)连续省略赋值
int a[][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

二维数组的遍历

```
#include <stdio.h>

int main()
{
    int grade[4][3] = {82,77,76,90,87,56,87,48,75,86,86,66};
    int i, j;
    for( i = 0; i < 4; i++ )
    {
        for( j = 0; j < 3; j++ )
        {
            if( grade[i][j] < 60 )
            {
                printf("第%d位学生不及格");
                continue;
            }
        }
    }
}
```

本讲小结

一、数组由类型说明符、数组名、数组长度（数组的个数）三部分组成 int a[10]; float money[2][3]; // 注意下标

二、数组的初始化 1)声明之后一个个赋值 2)省略形式赋值： char name[] = "Sam";

三、数组的遍历 通过while、do while、for等循环遍历

四、数组的作用 数组是程序设计中最常用的数据结构，属于构造类型。它将相同类型的数据连续组合在一起，简化了命名和赋值过程。相较单一的数据类型，能够更快更方便的处理更多的数据。

C语言入门教程 第8讲 指针的基本概念

指针的基本概念

“&” 与 “*”

取地址与指针运算符(取值运算)

“&” 对变量进行取地址运算: &(变量名) => 获取变量的地址 如：

```
int i;
printf("%d", &i);  // 打印出地址
```

如:

```
scanf("%d", &i);  // 传入一个地址
```

而“”作为指针运算符，当然我更乐意称它为取值运算：(变量名) => 将变量当做地址,到相应的地址取值

思考：

*(&(变量名)) 是什么？

```
int i = 6;
printf( "%d", *(&i) );
```

什么是指针变量？

指针变量就是用来存储地址的变量

任何变量都它存储的地址，这个地址是随机分配的。如：

```
int i = 10;
```

i 这个变量的地址，是系统随机分配的。而我们对其进行 “&” 运算之后就可以得到它的地址，并且 (&i) 这个表达式就是一个指向i的指针。

这个指针所指向的地址中存储的值则是： *(&i) ==> i

声明与初始化

声明方式：

指针类型 * 指针变量名;

例如：

```
int *p1;
char *name;
```

初始化：

1)普通写法

```
int x;
int *p;
p = &x;
```

2)省略写法

```
int x;
int *p = &x;
```

上述两种写法都声明了一个指针p,并且这个指针是指向变量 x 的 对于变量 x, (&x) 则是一个指向x的指针 （指针存放的就是地址, 而&运算得到的地址表达式也就是指向其本身的指针） 并且通过 *(指针) 取到这个地址中的值

常见错误

请不要人为的指派地址：

```
// 错误1
int *p;
p = 10010;

// 错误2
int x = 20;
printf("%d", &(*x) );

// 正确形式：
int i, *p, *t;
p = &i;
t = p;
```

只能通过已有的变量传递

经典的错误 - Scanf函数

```c
int score;
printf("请输入你的分数：\n");
scanf("%d", score);
```

上述代码在编译时没有错，运行时会出现严重的错误

错误原因： scanf() 函数后面的参数应该传入的是指针，而这里直接把 score 的值传了进去。

正确应该是：

```c
int score;
printf("请输入你的分数：\n");
scanf("%d", &score);
```

Swap交换两个变量

与第五讲中不同的是,使用指针到某个地址中取访问它的值,就没有了作用域的限制

```c
#include<stdio.h>

void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    printf("x=%d, y=%d \n", *x, *y);
}
int main()
{
    int i = 13, j = 45;
    swap(&i, &j);
    printf("i=%d, j=%d\n", i, j);
}
```

本讲小结

1) “&” 和 “*”

取地址与取值运算符 ==> 互为逆运算

2)指针变量是什么?

指针变量就是用来存储地址的变量 如,&i 就是一个指向变量i 的指针 通过*(指针)取值

3)声明

int *p1; char* name;

4)初始化

普通写法

```
int x;
int *p;
p = &x;
```

省略写法

```
int x;
int *p = &x;
```

5)指针的作用 引用类型，传递地址，减少内存消耗

课后作业

现在有一个数组，存储的是一个同学的期中考试成绩。

```
int score[8] = { 75, 86, 70, 88, 62, 87, 69, 77 };
```

那么现在我们要做的事情是：

1)求总分，求平均分

2)用指针遍历数组，求最大值和最小值

附加题： 用一个二维数组存储八门课的名称，例如：

```
char course[8][256] = { "chinese", "English", ... }
```

再用二维数组，存储多个人的成绩，用指针遍历求出每门课的平均分

C语言入门教程 第9讲 指针与数组

指针与数组

上节课后作业

现在有一个数组，存储的是一个同学的期中考试成绩。

int score[8] = { 75, 86, 70, 88, 62, 87, 69, 77 };

那么现在我们要做的事情是：

1)求总分，求平均分

2)用指针遍历数组，求最大值和最小值

附加题： 用一个二维数组存储八门课的名称，例如： char course[8][256] = { "chinese", "English" ... } 再用二维数组，存储多个人的成绩，用指针遍历求出每门课的平均分


```c
#include<stdio.h>
#include<stdlib.h>

int main()
{
    // 现在有一个数组，存储的是一个同学的期中考试成绩。
    char course[8][256] = { "语文", "数学", "英语", "物理", "化学", "生物", "历史", "地理"
};
    char name[5][256] = {"Alan", "Bob", "Clain", "David", "Elis"};
    int score[5][8];
    int sum[5] = {0};
    float avg[8] = {0};

    int i,j;

    for(i = 0; i < 5; i++)
    {
        for(j = 0; j < 8; j++)
        {
            score[i][j] = rand() % 100;
        }
    }


    for(i = 0; i < 5; i++)
    {
        for(j = 0; j < 8; j++)
        {
            sum[i] += score[i][j];
            avg[j] += score[i][j];
        }
    }


    for(i = 0; i < 8; i++)
    {
        printf("\t%s", course[i]);
    }

    printf("\t总分");

    for(i = 0; i < 5; i++)
    {
        printf("\n%s", name[i]);
        for(j = 0; j < 8; j++)
        {
            printf("\t %d",score[i][j]);
        }
        printf("\t%d", sum[i]);
    }

    printf("\n平均分");

    for(j = 0; j < 8; j++)
    {
        printf("\t%.1f", avg[j] / 8.0);
    }

    printf("\n");
    return 0;
}
```

字符数组与指针

字符串 == 字符数组+'\0'结尾

```
// 对于字符串"Hello world"相当于如下的字符数组
char hi[] = {'h','e','l','l','o',' ','w','o','r','l','d', '\0'};

printf("%c \n", hi[1]);  // e
printf("%c \n", "Hello world"[1]); // e
```

因为字符串相当于字符数组，所以"Hello world"[1]越数组hi[1]的值一样。

数组名 == 起始地址

学会自己研究各种变量

1)如何研究？ 学会善用printf

2)怎么知道变量是不是一个指针(或地址) 学会使用*

```
int a[5] = { 1, 3, 5 };

printf("%d",     a);    // 1638196
printf("%d",    *a);    // 1
printf("%d", *(a+1)); // 3
```

运行成功，表示这个变量的类型即使不是指针，也是一个地址

常见用法：

```
int a[5] = { 1, 3, 5 };
int *p;
p = a;
printf("%d", *(p+1)); // 3
printf("%d", *(p+2)); // 5
```

"Hello world"究竟是？

有了上面的两个结论之后，我就可以继续使用printf来探讨更多的东西。 比如"Hello world"这个字符串所代表的意义。 我们已经知道这个字符串实际上就是一个字符数组，通过"Hello world"[1]甚至可以取到其中的值。那么对于这样一个字符串而言它的数组名(地址)是什么？ 运用以上相同的方法可以看到：

```
printf("%d",   "Hello world" );   // 4337572
printf("%d", *("Hello world"));   // 72
printf("%c", *("Hello world"));   // H
printf("%c", *("Hello world"+1)); // e
```

多个名字如何用char数组存储？

多个人的名字：

```
char a[256] = "Alan";
char b[256] = "Bob";
char c[256] = "Cici";
```

现在要用数组来存储这三个人的名字。 也就是要用一个数组来存储三个数组，数组的每一个元素是一个数组：

```
char name[3][256] = {"Alan", "Bob", "Cici" };
```

如何使用指针遍历

```
int arr[8] = { 1, 1, 2, 3, 5, 8, 13, 21 };
int i;
int *p;
// 普通遍历
for(i = 0; i < 8; i++)
{
    printf("%d ", arr[i]);
}
// 指针遍历
for(p = &(arr[0]); p <= &(arr[7]); p++)
{
    printf("%d ", *p);
}
// 数组名获取指针地址
for(p = arr; p < (arr+8); p++)
{
    printf("%d ", *p);
}
```

本讲小结

1)如何使用指针遍历数组？ 使用循环

2)如何研究各种变量？ 尝试使用printf

3)数组与指针什么关系？数组名就是一个地址，指向这个数据的开端

```
int a[] = {1, 3, 5, 7, 9};
int *p = a;

printf( "%d \n",      *p == a[0] ); // 1
printf( "%d \n", *(p+1) == a[1] ); // 1
printf( "%d \n", *(p+2) == a[2] ); // 1
```

可以知道：

```
*(p+n) == a[n]
```

引申推导：

```
a[n] = *(p+n)
     = *(n+p)
     = n[a]
```

即：

```
a[n] = n[a]
```

可以这样理解，下标对于数组来说，就是相对于起始地址的一个偏移量。

字符串处理

字符与字符串的区别

1)符号

```
字符      ' ' 单引号
字符串     " " 双引号
```

2)内容

```
字符      单个字符
字符串     多个字符
```

3)输入输出格式

```
字符      %c
字符串     %s
```

4)标识

```
字符串     '\0'做结尾
```

字符串的本质是?

字符串本质上就是一个字符数组

"hello world" 这样的字符串，就是包含其中各个元素的，并且结尾为'\0'的一个一维字符数组

如:

```
char hi[12]={'H','e','l','l','o',' ','w','o','r','l','d','\0'};
```

因为一个字符串要有'\0'作为结尾 所以字符数组的长度要比本身字符数目多一个

如上例中,字符串有11个,但是字符数组长度要达到12

字符串的定义与初始化

字符串面量: "Hello world" 注意：字面量(双引号引起来的)是常量

区别举例：

```
"Alan"  常量
char name[20] = "Jack";   // "Jack"是字面量 是常量  name是字符数组, 是变量
```

注： 常量不能被赋值，如讲2的值赋给1这样：

```
1 = 2 是不行的，1是常量不能被赋值
```

"string" 这样的字符串也是不能被赋值的：

```
"string" = "Hello Jack" 这样是错的
```

字符数组

1)数组形式

```
char hi[3]={'H', 'i','\0'};
char hello[]={'H','e','l','l','o',' ','w','o','r','l','d','\0'};
```

2)直接赋首地址

```
char shool[20] = "第一中学";
```

3)省略长度

```
char name1[] = "Alan";
char name2[] = { "Alan" };
```

字符指针

```
char *name3  = "Alan";
char *s = "第一中学";
```

思考: 指针能不能直接赋给数组?

字符串的遍历

```
char hi[] = "I am happy" ;
```

1.for循环遍历

```
for(i=0; i<11; i++)
{
    printf("%c", hi);
}
```

2.直接输出

```
printf(hi);
```

3.字符串格式(%s)输出

```
printf("%s", hi);
```

字符串数组（二维字符数组）

常见姓名：

```
char name1[] = "张三";
char name2[] = "李四";
char name3[] = "王五";
```

存储多个字符串：

1)二维数组

```
char name[][256] = { "张三", "李四", "王五" };
```

2)一维指针数组

```
char *name[] = { "张三", "李四", "王五" };
```

对于printf，最重要的是？

对于printf和scanf来说，字符串最重要的是其起始地址

如打印一个字符串：

```
char *text = "hey gays!";

1) pirntf(text);
2) printf("%s", text);
```

对于printf这个函数而言，只要知道字符串开头的地址， 那么就一直打印下去，一直到碰到'\0'的时候停止

如：

```
printf("Hello \0 world"); \\ hello
```

常见的字符串操作函数

stdio.h

输入：scanf()、gets()、getchar() 输出：printf()、puts()、putchar()

string.h

连接：strcat() 拷贝：strcpy() 比较：strcmp() 获取长度：strlen() 大小写转换：strupr()、strlwr() 拼接字符串：sprintf()

示例：

```
#include<stdio.h>
#include<string.h>
int main()
{
    char *hi = "hello", s2 = "Alice";
    char str[256], *pstr;
    printf("%s \n", strcat(hi, " world"));//连接
    printf("%s \n", strcpy(str, "come on")); //拷贝
    printf("%s \n", strcmp("你好", "你好")); //比较
    printf("%s \n", strlen(str)); //获取长度
    printf("%s \n", strupr(str)); //转成大写
}
```

学会查询文档：

1.右键查看定义 2.度娘谷歌 3.手册 4.自己撰写自己的文档

关于函数传参数

```
对于：int add(int a, int b);
调用：add(5, 10);

add函数中参数初始化时，相当于：
int a = 5;  int b = 10

对于：int swap(int *x, int *y);
调用：swap(&i, &j);

swap函数中参数初始化时 ，相当与：
int *x = &i;  int *y = &j;
```

strlen()

```
#include<stdio.h>
int mylength(char *aim);

int main()
{
    char *str1 = "hello ";
    printf("%d\n", mylength(str1));
}

int mylength(char *aim)
{
    int count=0;
    while(*aim++){
    count++;
    }
    return count;
}
```

strcat()

```
#include<stdio.h>
char * mycat(char *dest, char *src);

int main()
{
    char str1[20] = "hello "; // 要拷贝的话需要有足够的内存
    char *str2 = "world";
    printf("%s\n", mycat(str1, str2));
}

char *mycat(char *dest, const char *src)
{
    char *address = dest;

    while (*++dest) // 加到 '\0' 马上停止
        ;

    while (*dest++ = *src++)
        ;

    return address;
}
```

(感谢 ...蕾依丽雅 君的提醒, 该函数已修正)

本讲小结

一、字符串的本质，与字符的区别 1)本质上是结尾为'\0'字符数组 2)单双引号，书写区别 3)输出格式"%c"和"%s"

二、字符串的定义与初始化 1)字面量 2)数组 3)指针

三、常见的字符串操作，及其操作的实现 连接(strcat)、拷贝(strcpy)、比较(strcmp)、获取长度(strlen)、大小写转换(strupr、strlwr)、拼接字符串(sprintf)

以上部分函数的实现跨越在博主的 string.h 的实现中找到 博主的 C标准库

课后作业

通过自己查找参考，理解并实现以上常见的字符串操作 注：sprintf()只要掌握，不要求实现

调试以及资料收集

视频地址

A站：http://www.acfun.tv/v/ac479828 B站：http://www.bilibili.tv/video/av400594/ 新浪：http://video.sina.com.cn/v/b/90825190-2731231323.html

如何进行调试

1)注释调试 2)printf打印调试 3)根据报错信息找错 4)使用调试工具 5)求助网络

查询文档

1.右键查看定义 2.度娘谷歌 3.手册 4.自己撰写自己的文档

如何查看函数以及变量的引用？ 使用工具自带的功能。

如何查看函数的使用方法？ 度娘谷歌等，vs下可按F1查看微软的帮助文档

基础的东西学完了如果知道下一步学什么？ 通过相关的书籍、论坛帖子或者博客博文

提高自己的查找能力

查询： http://www.baidu.com/ 度娘 http://www.google.com.hk/ 谷歌 http://msdn.microsoft.com/ MSDN

百度知道、各大博客论坛的搜索功能

资料网站： http://www.verycd.com 电驴 http://www.51cto.com 51CTO http://club.topsage.com/ 大家网 http://www.oschina.net/ 开源中国

论坛： http://bbs.chinaunix.net/ ChinaUnix论坛 http://bbs.51cto.com/ 51cto论坛 http://www.itpub.net/ itpub论坛 http://bbs.pediy.com/ 看雪学院 http://www.cctry.com/ vc驿站 http://bbs.fishc.com/ 鱼C论坛 http://bbs.bccn.net/ 编程论坛 http://www.rupeng.com/ 如鹏网

技术博客聚集区： http://blogs.msdn.com/ MSDN博客 http://www.cnblogs.com/ 博客园 http://blog.51cto.com/ 51cto博客 http://www.iteye.com/blogs iteye博客 http://space.itpub.net/ itpub博客 http://wo.zdnet.com.cn/ ZDNet技术社区 http://www.oschina.net/blog 开源中国

英文网站： http://sourceforge.net/ http://www.cplusplus.com http://www.codeguru.com/ http://www.codeproject.com/

书籍教程推荐

《C语言自学教程》郝斌 《数据结构自学视频》郝斌 《C Primer Plus》中文版 《C语言程序设计 现代方法》 《狂人C-程序员入门必备》 《C和指针》人民邮电出版社 《C语言实例解析精粹》曹衍龙

如何学习？

1.理解编程过程中的知识点 2.使用自己理解知识去编程、在编程的过程中验证所学 3.尝试去寻找题目编写 4.在调试中寻找自己的不足 5.阅读好的书籍，查漏补缺(可搜索笔记)

小结

1.调试的基本方法 2.去哪里找资料？ 3.如何学习比较好？

如果有发现不错的网站或者资料可以向博主推荐 特别是各个博客区中比较好的博客 后期还会再排版，汇总推荐

C/C++ Beginner 一群 10191598 C/C++ Beginner 二群 163859361 （感谢One day捐助） C/C++ Beginner 三群 10366953 （感谢Vol jun捐助）

c语言入门教程 第12讲 结构体

结构体

结构体

可以将多种数据类型组合起来的结构

声明方式

struct 结构体名称{ 成员1的类型 成员1的名称; 成员2的类型 成员2的名称; ...... 成员3的类型 成员3的名称; } ;

举例：

```
struct time{
    int hour;
    int minute;
    int second;
} ;
```

注意不要漏掉了最后的分号

结构体的定义

1)常规定义

```
struct time{
    int hour;
    int minute;
    int second;
} ;

struct time t;
```

```
如何定义一个int类型的变量？
```

```
    int number;
```

实际上来说，结构体变量的定义是跟普通数据类型类似的。 如，以上两个变量的声明区别，仅在于，一个的类型是struct time 另一个的类型是int

2)声明的同时定义

```
struct student{
    char name[256];
    char sex[2];
    int age;
    int grade;
} Alan, Tom ;
```

3)使用结构体作为成员

```
struct DATE{
    int year;
    int month;
    int day;
} ;

struct person{
    char name[256];
    struct DATE birthday;
} ;

struct time t;
```

4)匿名结构体

```
struct {
    int number;
    char name[256];
    char sex[2];
    int age;
    int grade;
} Alan, Tom ;
```

结构体的引用与初始化

使用“.”成员运算符来获取结构体中的成员

```
#include<stdio.h>#include<string.h>

struct student{
    int number;
    char name[256];
    char sex[2];
    int age;
    int grade;
} ;
int main()
{
    struct student alan;
    alan.number = 001;
    strcpy(alan.name, "Allan");
    strcpy(alan.sex, "男");
    alan.age = 18;
    alan.grade = 3;

    printf("学号：%d \n姓名: %s \n性别：%s \n年龄：%d\n年级：%d\n",
           alan.number, alan.name, alan.sex, alan.age, alan.grade);
}
```

结构体数组

结构体数组的使用：

```
#include<stdio.h>
#include<string.h>

struct student{
    int number;
    char name[256];
    int age;
} ;
int main()
{
    struct student class_02[5];
    int k;

    for(k=0; k < 5; k++)
    {
        printf("Please input number:\n");
        scanf("%d", &class_02[k] .number);
        printf("Please input name:\n");
        scanf("%s", class_02[k] .name);

        printf("Please input age:\n");
        scanf("%d", &class_02[k] .age);

        printf("\n");
    }

    printf("Num\tName\tage\n");

    for(k=0; k < 5; k++)
    {
        printf("%d\t", class_02[k] .number);
        printf("%s\t", class_02[k] .name);
        printf("%d\n", class_02[k] .age);
    }
}
```

```
#include<stdio.h>

struct student{
    int number;
    char name[256];
    char sex[6];
    int age;
    int grade;
} ;
int main()
{
    struct student class1[5] = {
        { 01, "Alan", "man", 16, 12 },
        { 02, "Bob", "man", 18, 12 },
        { 03, "Cici", "woman", 16, 12 },
        { 04, "David", "man", 11, 12 },
        { 05, "Elis", "woman", 16, 12 },
    };
    for(k = 0; k < 5; k++)
    {
        printf("\nstudy number: %d \nname: %s \nsex: %s \nage: %d\n grade: %d\n" ,clas
s1[k].number,
        class1[k].name,
        class1[k].sex,
        class1[k].age,
        class1[k].grade);
    }
}
```

结构体指针

使用分量运算符“->”来获取成员

```
#include<stdio.h>

struct DATE{
    int year;
    int month;
    int day;
} *date ;
int main()
{
    date->year = 2012;
    printf("%d", date->year);
}
```

```
#include<stdio.h>

struct DATE{
    int year;
    int month;
    int day;
} date = { 2012, 11, 27 }, *d ;
int main()
{
    d = &date;
    printf("%d", d->year);
}
```

用于函数的参数以及返回值:

```
#include<stdio.h>

struct time add(struct time now, struct time pass);

struct time{
    int hour;
    int min;
} ;
int main()
{
    struct time now = { 3, 55 }, pass = { 1, 33 }, result;
    result = add(now, pass);
    printf("%d:%d\n", result.hour, result.min);
}

struct time add(struct time now, struct time pass)
{
    struct time rel;
    rel.hour = now.hour + pass.hour + (now.min + pass.min)/60 ;
    rel.min = (now.min + pass.min)%60;
    return rel;
}
```

信息存储 3.1 计算机数据存储单位 3.2 进制转换 3.2.1 常见的进制转换 3.2.2 二进制简介 3.2.3 进制计算的基本概念 3.2.3.1 七进制计算 3.2.3.2 六十进制计算 3.2.3.3 十进制计算 3.2.3.4 进制计算的规律与公式推导 3.2.4 二进制与十进制的转换 3.2.4.1 二进制转十进制 3.2.4.2 十进制转二进制 3.2.4.2.1 六十进制中 3.2.4.2.2 十进制中 3.2.4.2.3 除二取余法原理 3.2.5 二进制与十六进制的转换 3.2.6 大端法与小端法机器 3.3 数字与编码 3.3.1 ASCII编码 3.3.2 数字的计算 3.3.3 计算机中的数字与编码对比

信息存储

1. 计算机数据存储单位
2. 进制转换
    - 常见的进制转换
    - 二进制简介
    - 进制计算的基本概念
        - 七进制计算
        - 六十进制计算
        - 十进制计算
        - 进制计算的规律与公式推导
    - 二进制与十进制的转换
        - 二进制转十进制
        - 十进制转二进制
            - 六十进制中
            - 十进制中
            - 除二取余法原理
    - 二进制与十六进制的转换
    - 大端法与小端法机器
3. 数字与编码
    - ASCII编码
    - 数字的计算
    - 计算机中的数字与编码对比

计算机数据存储单位

```
1TB (Trillionbyte 万亿字节 太字节 )      = 1024 GB
1GB (  Gigabyte   吉字节 又称“千兆” ) = 1024 MB
1MB (  Megabyte   兆字节 简称“兆” )   = 1024 KB
1KB (  Kilobyte   千字节 )              = 1024 B
1B  (    Byte     字节 )                 =   8  Bit
```

各位需要注意，内存上最小的单位是字节（byte）没有比这更小的单元了。而我们讨论的位（bit）是比字节还要小的单位，这个单位在计算机上是无法直接查看的（因为字节已经是最小的了），但是实际上而言计算机真正的所有数据都是有一个位一个位的1010之类的二进制数据组成的。而我们要进行C/C++编程的话，很多时候都需要去处理这个比内存最小单位还要小的位，如果是专做底层的话使用位操作就更加频繁了，所以我们必须要了【位级】也就是【二进制层面】的数据形式。

进制转换

常见的进制转换

生活中常见的进制：}

我们的生活中充斥着各种各样的进制转换，例如每天工作生活的时候，可能很多人都不知道今天几号，但是却一定知道今天星期几，这个星期的概念中就存在着进制转换。满了七天就可以进一位变成一个星期，这是一个很常见的进制转换。

上述还列举了好几种其他的进制转换，我们可以很清晰的发现，其实不同的进制在【进位】的时候很明显的体现出一个规律便是：几进制就满几进一。

二进制简介

那么实际上二进制跟上述常见的进制都是类似的，也就很简单的是满 2 进一位，我们可以看如下的式子：

```
1 + 1 = 2
```

这在我们十进制中是一个幼儿园的小孩子都懂的加法，但是到了二进制中这个结果会变得不一样了：

```
1 + 1 = 10
```

乍一看会觉得很难接受，但是仔细一想这也是可以理解的，因为在二进制中 满2 就会进一位，所以 1+1=2 已经满2了就进了一位结果就是 10 了。不过各位需要注意的是这个 10 是二进制中的 10 ：

```
10(二进制) = 2(十进制)
```

那么我们的二进制，再加几个数字看看:

```
10 + 1 =  11 （等于十进制中的3）
11 + 1 = 100 （等于十进制中的4）
```

情况就是这样，我们可以用这样的方式依次类推下来：

```
二进制    十进制
     1        1
    10        2
    11        3
   100        4
   101        5
   110        6
   111        7
  1000        8
  1001        9
  1010       10
  1011       11
  1100       12
  1101       13
  1110       14
  1111       15
 10000       16
```

有些人可能会觉得奇怪，为了为什么这个看起来没有开始七进制那个星期的那样好理解，这里博主解释一下。

真正数学意义上的七进制里面是没有7的，因为到了6之后再加一 满 7 就要进一位变成 10 ，这个形式我们不是很熟悉但是换个熟悉的方式就理解了，因为我们平常生活着中把这个七进制的【个位】叫做【天】，【十位】叫做【星期】所以我们的七进制的：

```
10 天
```

也可以看做

```
1星期 0天
```

进制计算的基本概念

首先我们要了解的是进制本身的一个概念：

七进制计算

以星期的七进制举例：

```
21
```

这个值代表的天数到底是多少？我们将其换成我们所熟悉的单位一看：

```
2 星期 1 天
```

这样很直观的就能算出来 这个七进制的 21 所代表的的是十进制的 15 天，为什么呢？

很简单，因为一个星期是7天，所以上述的七进制数字转成十进制就是：

```
7*2 + 1 = 15
```

六十进制计算

以时分秒的六十进制举例：

```
356
```

那么这个六十进制的 356 秒到底是十进制的多少秒？乍一看又有点晕了，不过没关系，换成我们所熟悉的单位来看看：

```
3小时 5分钟 6秒
```

这样看一下一下子就觉得简单了，我们也可以开始算了，答案就是：

```
3*3600 + 5*60 + 6 = 11106（秒）
```

十进制计算

接下来我们再用十进制来举例：

十进制的：

```
1234
```

我想这个大家应该一眼就能看出来这个十进制的值，那么我们要看的是这个十进制的每一位拆开来看的式子：

```
1*1000 + 2*100 + 3*10 + 4
```

进制计算的规律与公式推导

相信经过上面三种进制的举例之后，大家的心里已经模糊的有点想法，接下来我们就要把这个模糊的概念变的清晰。

首先我们定义一个未知数x（x>=2），那么假设我们所使用的这个进制是 x进制，那么这个 x 进制有什么特点？

不用多说：

```
① 满x进一
```

相信大家可以理解

那么对于这个 x进制 的个位，单位的话一定 1，因为满了 x 就会进一位，所以由 ① 式可得：

```
② 个位的最大值是 x-1
```

那么各位同学想想，由 ①、② 式可以知道什么？那么博主也不卖关子：

```
③ 第二位的 1 是第一位的 x 倍
```

那么接下来的问题就是：

```
x进制的 abcd （abdc 分别是0到x-1的数字） 等于 10进制的多少？
```

大家可以停在这里先仔细考虑这个问题，当然草稿纸也是必要的，因为经过思考之后再来看博主的推理绝对会效果更佳！

那么，如果你已经思考过了我们就继续往下看。

由 第①、③式 就可以知道我们想要的结论了：

关于x进制的 abcd：

```
个位数就是   d
十位数是     x*c
百位数是   x*x*b
千位数是 x*x*x*a
```

所以最后答案是：

```
④ 1111（x进制）= a*x^3 + b*x^2 + c*x^1 + d*x^0 （10进制）
```

备注：

1. x^3 的意思就是 x的3次方 即 xxx
2. x^0 的意思是 x的0次方 即 1（除0以外，任意数的0次方都是1）

那么这个式子 ④ 是一个通用形的公式，可以计算任意进制转十进制。各位可以摊开草稿纸自行推导，以及套用求，博主这里就不废话了。

二进制与十进制的转换

二进制转十进制

既然有了公式，那么二进制转十进制自然是可以套用通用公式来计算的：

```
例如 0100 1011 的十进制计算：

0100 1011(二进制) = 1*2^6 + 1*2^3 + 1*2^1 + 1*2^0 = 75 (十进制)
```

或者使用 1248 法，那么什么是 1248 法，额，博主在写的时候才发现其实这个就是公式一种运用方式吧。就是记住2 的各个次方，依次从0次方开始分别是：1、2、4、8、16、32、64...... 等等

然后计算的时候碰到如下情况，直接套用：

```
如 0001 0101 的十进制值计算：

0001 0101(二进制) = 16 + 4 + 1 = 21
```

二进制的话，0就是没有值不用乘，1的话乘的时候可以省略，所以只要记住一些常见的 2 的次方就可以快速的将二进制转成十进制了。博主在视频当中二进制转十进制的时候就是这么做的 =v=

十进制转二进制

使用【除二取余法】，例如：

```
789(十进制) = 1100010101(二进制)

 789 / 2 = 商 394 余 1  个位
 394 / 2 = 商 197 余 0  十位
 197 / 2 = 商 98  余 1  百位
  98 / 2 = 商 49  余 0  第..位
  49 / 2 = 商 24  余 1  第..位
  24 / 2 = 商 12  余 0  第..位
  12 / 2 = 商 6   余 0  第..位
   6 / 2 = 商 3   余 0  倒数第3位
   3 / 2 = 商 1   余 1  倒数第2位
   1 / 2 = 商 0   余 1  倒数第1位
```

那么我们的余数倒过来就是我们的二进制的值了。

六十进制中

11106 秒转 六十进制

```
11106 / 60 = 商 185  余数 6
  185 / 60 = 商 3    余数 5
    3 / 60 = 商 0    余数 3
```

所以 11106 = 3小时 5分 6秒

十进制中

```
1234 / 10 = 商 123 余 4
 123 / 10 = 商 12  余 3
  12 / 10 = 商 1   余 2
   1 / 10 = 商 0   余 1
```

除二取余法原理

设两个未知数 x, y (x>=2) ，假设我们面对的是 x进制，要把十进制数 y 转成 x 进制

那么我们一定要清楚的是这个 y 去除以 x 能得到什么

假设这个 x进制的数是 abcd (abcd为0~x之间常数) 当然这个x进制的数可能没有4位也可能超过4位，这没有关系我们只是看个示意，那么有了这个示例，根据我们最开始算出来公式可以得出下列等式：

```
abcd(x进制) = a*x^3 + b*x^2 + c*x^1 + d*x^0 (十进制) = y
```

用这个数去除以 x 你会发现：

```
y / x = ( a*x^3 + b*x^2 + c*x^1 + d*x^0 ) / x (十进制)
      = 商 a*x^2 + b*x^1 + c*x 余数 d
```

那么获取的这个余数 d 就是我们 abcd 这个 x进制数的【个位】也就是第一位

```
( a*x^2 + b*x^1 + c*x ) / x = 商 a*x^1 + b*x 余数 c
```

再用这个商去除以 x 就能获得 x进制数的 第二位。依次类推，只要商不为0，我们就可以通过这个办法一直求下去。

各位可以用摊开草稿纸残念一下，也可以用这个结论反过来看看前几个 二进制、十进制、六十进制 用【除二取余法】的数据，博主就不赘述了，顺便感慨一下，百度百科上面那些进制转换的原理好残念，完全看不懂，不知道写着有什么意思。

二进制与十六进制的转换

如果上面的公式你都有参与推倒，并且有使用公式进行过一定的转换联系，那么这个部分的内容对你而言肯定是小case

```
  二进制     十六进制  十进制
 0000 0000    00       0
 0000 0001    01       1
 0000 0010    02       2
 0000 0011    03       3
 0000 0100    04       4
 0000 0101    05       5
 0000 0110    06       6
 0000 0111    07       7
 0000 1000    08       8
 0000 1001    09       9
 0000 1010    0A       10
 0000 1011    0B       11
 0000 1100    0C       12
 0000 1101    0D       13
 0000 1110    0E       14
 0000 1111    0F       15
 0001 0000    10       16
 0001 0001    11       17
 0001 0010    12       18
 0001 0011    13       19
 0001 0100    14       20
 0001 0101    15       21
 0001 0110    16       22
 0001 0111    17       23
 0001 1000    18       24
 0001 1001    19       25
 0001 1010    1A       26
 0001 1011    1B       27
 0001 1100    1C       28
 0001 1101    1D       29
 0001 1110    1E       30
 0001 1111    1F       31
 0010 0000    20       32
```

简单的观察一下就会发现上述二进制的低四位对应十六进制的第一位，上述二进制的高四位对应十六进制的第二位。

这其实是一个很简单的事情，因为【一】个【十六进制】的位刚好包含了【四】个【二进制】的位，这是因为十六进制的 x 是 16也就是 2的四次方，而二进制的 x 是 2 即 2的1次方。

我们通过观察上述数据也可以发现二进制的 0~1111 对应的刚好就是 十六进制的 0 ~ F 也就是二进制的四个位刚好对应一个十六进制的位。所以二进制与十六进制的转化十分简单：

```
二进制    ：1111 0001 1010 0101 0111 1000
十六进制  :    F    1    C    5    7    8
```

大端法与小端法机器

大端法和小端法，指的是数据的两种存储方式，对于一个数据 0x01234567 而言在

大端法机器上是

```
      低 ------------------------> 高
内存地址：0x100 0x101 0x102 0x103
             01    23    45    67
```

小端法机器上是

```
      低 ------------------------> 高
内存地址：0x100 0x101 0x102 0x103
             67    45    23    01
```

这是两种计算机存储字节数据的方式（2个十六进制的数表示8个二进制的位也就是1个字节）。大端法和小端法分别代表两种思想，一种是亲【人类直观】另一种是倾向【数学意义】

最开始看的时候会举得小端法有些不科学，其实这个是比较符合【数学意义】的，因为 67 是数字 0x01234567 中的个位与十位，是最低的一个字节，那么按照数学意义上来说的话，存储数据的时候从最低的字节开始写入是一种很正常的想法。只不过看起来不是那么直观。

而大端法的话，把最高字节的数据放在内存的低位其实就【数学意义】上看来是有些奇怪的，不过我们作为人看来却是比较亲切。

二者之间各有优点，各位只要了解即可，这个大小端只是做一个基础知识点。实际上编程时很少会感受到这中间的差别。

最后附上博主查看大小端的代码：

#include // 包含 stdio 这个库的头文件 /* * main 函数 是程序的入口 * * int 是程序的返回值类型 * main 是函数名 * int argc 是参数个数 * char const *argv[] 是指向参数的指针数组 */ int main(int argc, char const *argv[]) { int i = 0x01234567; char *p; p = (char *)&i; printf("%x ", *(p+0)); printf("%x ", *(p+1)); printf("%x ", *(p+2)); printf("%x ", *(p+3)); // main 函数的返回值 // return 是返回的关键字 return 0; }
由于大部分语法尚未开始讲解，所以这个程序的具体含义各位可以留待以后再来探讨。

数字与编码

ASCII编码

在数据的世界里，实际上是没有任何中英文之类的字符. 所有的数据全都是1010之类的二进制组成的数字. 那么在这样的情况下，我们要在计算机上使用字符就需要使用硬件上自带的编码才能把这些数字的数据编程我们所需要的文字等字符. 以下是使用范围最广的 ASCII 编码的一部分:

二进制	十进制	十六进制	图形
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W

0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

完整请查看: http://zh.wikipedia.org/wiki/ASCII

ASCII（'''A'''merican '''S'''tandard '''C'''ode for '''I'''nformation '''I'''nterchange，'''美国信息交换标准代码'''）是基于拉丁字母的一套电脑编码系统, 它主要用于显示现代英语.

对于 一个数字 65 如果我们告诉计算机我们要将其当做字符来使用, 那么这个数据就会被转换成对应的字符 '''A'''. 当然用 'A' 这样的字符也可以当做数字来计算, 因为虽然看起来它是个字符, 但实际上内存中只会存储 65 这个数字, 所以 'A' + 1 是可以计算的, 而且结果是 66, 转成字符就是 'B'.

数字的计算

数字的计算跟平常数学意义上的计算没有太大的差别, 不过编程中的计算会有某些方便的拓展, 如位运算. 具体细节请参见后面的 CPU 计算章节.

计算机中的数字与编码对比

在计算机当中, 所有的数据都是 1010 的二进制形式, 当你把这个数据当成是字符的时候, 计算机就会查找对应的编码, 当你把这个数据当成是图片时, 计算机就会按照对应的图片格式来加载这个数据. 如果仅仅是当做数字来计算的话, 那么就是普通的数学计算.

最开始学习编程基础的时候, 我们最需要了解的就是字符和数字的区别. 字符只是对应编码转换出来的一种符号, 如果编码不同的话字符转换出来的符号也可能不同, 所以并没有很强的数学意义. 而且在 C语言中 用于保存字符的 char 类型单位大小只有 1字节 也就说最多只有 255 种情况. 如果用字符来做计算的话也只能做一些简单的计算, 超过范围的话就会因为单位过小而无法保存运算的结果.

而在 C语言中 有 int 类型专门用于做整数的计算, 在 32 位操作系统上 int

CPU如何进行数学运算

1. 位运算
    - 与或非
        - 按位与
        - 按位或
        - 按位取反
    - 位移
        - 左移
        - 右移
        - 算术位移与逻辑位移
2. 四则运算
    - 加
    - 减
    - 乘
    - 除

位运算

与或非

按位与

符号: & 示例：

```
1(10进制)  =  0000 0001 (2进制)
9(10进制)  =  0000 1001 (2进制)
1  &  9     =  0000 0001 (2进制)
            =  1 (10进制)
```

计算遵守：都为眞则眞，除此之外为假

按位或

符号：| 示例：

```
5(10进制)  =  0000 0101 (2进制)
8(10进制)  =  0000 1000 (2进制)
5  |  8     =  0000 1101 (2进制)
            =  13 (10进制)
```

计算遵守：只有要一个是真结果就是真, 都为假才假

按位取反

符号：~ 示例：

```
5(10进制)  =  0000 0101 (2进制)
    ~5     =  1111 1010 (2进制)
           =  -6 (10进制)
```

计算遵守：真变假，假变真

PS：二进制负数的计算是

```
            0  =  0000 0000
 0 - 1  =  -1  =  1111 1111
-1 - 1  =  -2  =  1111 1110
-2 - 1  =  -3  =  1111 1101
-3 - 1  =  -4  =  1111 1100
-4 - 1  =  -5  =  1111 1011
-5 - 1  =  -6  =  1111 1010
```

位移

左移

符号：<< 运算：直接将数据向左移动一位，并补零

```
1 (10进制) = 0000 0001 (2进制)
1  <<  1   = 0000 0010 (2进制)
           = 2

1  <<  3   = 0000 1000 (2进制)
           = 8
```

右移

符号：>> 运算：直接将数据向左右动一位，并补零

```
9 (10进制) = 0000 1001 (2进制)
9  >>  1   = 0000 0100 (2进制)
           = 4

9  <<  3   = 0000 0001 (2进制)
           = 1
```

算术位移与逻辑位移

位移包括算术位移和逻辑位移两种情况。逻辑位移是指位移过后直接补零。

算术位移是指在保持符号位不变的情况下进行的位移。

例如：

```
-6 (10进制) = 1111 1010 (2进制)
-6  >>  1   = 1111 1101 (2进制)
            = -3

-6  >>  2   = 1111 1110 (2进制)
            = -1
```

对于 -6 而言最高位 1 是负数标识的符号位，在进行位移的时候该符号位会保持不变，这种位移叫做算术位移。而不论如何位移过后都补零的情况测试逻辑位移

四则运算

加

减

乘

除